Image Format Converter - RAW, JPG, PNG e TIFF

Project Implementation Report

Introduction

While using image editors, I noticed that many do not support **RAW** files or limit the number of images that can be converted for free.

The idea for this project came about with the goal of creating an efficient solution for bulk image conversion, capable of handling **RAW** files and other common formats (**JPG**, **PNG**, and **TIFF**), preserving quality and automating the process for entire folders of images.

The ultimate goal is to allow photographers, designers, and any user to convert and standardize large volumes of images quickly and without the need for paid software.

Initial Overview

The project arose from a personal need identified when using existing image editors. I found that, although there were high-quality editors available, none offered efficient support for bulk **RAW files**, and most online solutions had limitations in terms of quantity or required payment.

The first idea was to create a simple tool that would allow the conversion of **RAW** images to **JPEG**, focusing on basic functionality: select an image, process it, and save it in a new format. The initial goal was to demonstrate that batch conversion of **RAW** was possible using **Python** and specialized libraries, such as ***RawPy*** for **RAW** files and **Pillow** for common formats.

As the project progressed, new opportunities for improvement arose:

Extension to other formats

To make the project more universal, support for **PNG** and **TIFF** formats was added, allowing conversion between **JPEG**, **PNG**, **TIFF**, and **RAW**.

This required additional study of color modes and palette compatibility (**RGB**, **RGBA**, and **P**) to ensure that images did not lose important channels during conversion.

Batch processing

From the outset, it was essential that the tool be capable of processing **entire folders of images,** saving the user time and increasing the program's usefulness.

This led to the choice of the ***Pathlib*** library to iterate and manipulate files safely and efficiently.

Simple and functional graphical interface

I chose to use ***Tkinter***, despite its outdated appearance, due to its simplicity and compatibility with standard **Python**.

The idea was to create an intuitive window with:

- Button to select an input folder;
- Combo box to select the desired output format;
- Button to start conversion;
- Log area for real-time feedback.

Quality and compatibility

The priority was to maintain the quality of the converted images.

To this end, when converting **RAW**, I applied corrections such as **camera white balance**, **8-bit** per channel output, and **sRGB** color space.

In common formats, **color mode conversions** were performed when necessary (RGBA → RGB for JPEG, etc.) to avoid information loss or recording errors.

Modular planning

The project was structured in a modular way, **separating** the conversion, folder selection, and logging functions.

This approach allows for future expansions, such as support for new formats, image filters, or integration with parallel processing to speed up large batches.

In short, the initial vision evolved from a simple idea of **RAW** conversion to a universal and efficient converter capable of handling multiple formats, entire folders, and different color modes, while maintaining quality and ease of use for the end user.

Technical Concepts

RAW

This is a raw file generated directly by the camera sensor. It is uncompressed, which allows **all** the captured color and brightness details to be preserved.

Some of the advantages are maximum quality and flexibility for post-processing, as well as the possibility of making fine adjustments to exposure, white balance, and shadow/highlight recovery.

The disadvantages are that the files are very large and each camera manufacturer has specific extensions (**.cr3**, **.nef**, **.arw**, .**dng**), which creates a dependency on specialized libraries such as **RawPy**.

The color palette typically stores linear information from the sensor, which can later be converted to color spaces such as **sRGB** or **AdobeRGB**.

JPEG

This is a **lossy** compressed format, very popular in digital applications.

It offers advantages such as an **excellent ratio** between quality and file size. Highly compatible (**works on virtually any device**).

On the downside, compression **causes loss of detail**, especially in areas with smooth color transitions. Repeated conversions can further degrade quality.

The color palette is **RGB** only, with no support for transparency (**does not store the alpha channel**).

It is ideal for photographs and images where small file size is more important than maximum fidelity.

PNG

It is a **lossless** format with efficient compression for images with uniform color areas.

Among its advantages are transparency support through the ***Alpha*** channel (**RGBA**) and suitability for graphic images such as logos, illustrations, and UI visual elements.

The disadvantages are that files are larger than **JPEG** format and, for high-resolution photos, it is not the most efficient in terms of size.

The color palette can be **RGB** or **RGBA**, ensuring total color fidelity.

It is ideal for graphics, logos, and images that require transparency or total preservation of the original pixels.

TIFF

Versatile, lossless format widely used in printing, graphic design, and professional photography.

It has many advantages, but the most significant are the maintenance of maximum image quality and support for multiple layers, transparency, and different compression modes (including no compression).

On the other hand, the disadvantages are **extremely large files** and the fact that it is rarely used in everyday applications due to its size.

The color palette is **RGB**, **RGBA**, **CMYK** (for printing), and even high-precision color spaces (**16 bits per channel**).

The ideal use is as a master file for printing or professional image archiving.

NOTES

One of the technical challenges was ensuring color mode compatibility with the final format during image conversion.

- The **JPEG** format does not support **RGBA**, so it must be converted to **RGB**.
- **PNG** and **TIFF** formats can work with **RGBA**, preserving transparency.
- After post-processing, the **RAW** format is converted to **RGB** or **RGBA** in order to be saved in common formats.

This compatibility was handled in the code with conditionals that check the image mode (`image.mode`) and apply the necessary conversions (`.convert("RGB")` or `.convert("RGBA")`).

Libraries Used

1. Pillow

Pillow is the continuation of the old **Python** image library (**PIL**). **It is currently the most popular and established library for image manipulation in Python.**

It was used in this project because of its robust support for common formats such as **JPEG**, **PNG**, and **TIFF**, its ability to convert color modes, resize images, apply filters, and save in different formats, as well as its simplicity and efficiency in bulk image manipulation.

Some of the functions used were:

`Image.open()` - Opens an image and loads it as a manipulable object.

`Image.convert()` - Converts the image mode (e.g., **RGBA** → **RGB**, required for **JPEG** compatibility).

`Image.save()` - Save the image in a specific format, with options such as quality and compression.

```
imagem.save(caminho_imagem, format = formato_desejado, quality = 95)
```

Image 1 - Examples of using the ***Pillow*** library

```
imagem_raw = Image.fromarray(RGB)
```

Image 2 - Examples of using the ***Pillow*** library

2. RawPy

Library specialized in reading and processing images in **RAW** format. It is based on the powerful ***LibRaw library***, which interprets files produced directly by digital camera sensors.

Each camera manufacturer (**Canon**, **Nikon**, **Sony**, etc.) creates a specific type of **RAW** file (**.cr3**, **.nef**, **.arw**, **.dng**). **RawPy** allows you to read all these formats without relying on proprietary software.

It also allows for controlled post-processing: white balance, gamma curve application, bit depth, etc.

Some of the functions used were:

`rawpy.imread()` - Reads a **RAW** image and creates a manipulable object.

`.postprocess()` - Converts a **RAW** file into a **NumPy array**, in a defined color space (e.g., **sRGB**).

```
with RAW.imread(str(ficheiro)) as raw:
    RGB = raw.postprocess(
```

Image 3 - Example of using the ***RawPy*** library

3. Tkinter

Python's standard library for building graphical user interfaces (**GUIs**). Despite its somewhat outdated appearance, it remains widely used due to its lightness and direct integration with **Python**, without the need for external packages.

In this project, it was used because it allows you to create an intuitive interface where the user can select a folder, choose the output format, and start the conversion without having to interact with the terminal.

It includes basic graphical elements such as buttons, checkboxes, text boxes, and dialog boxes for folder selection.

Some of the functions used were:

`ttk.Combobox()` - ComboBox used to select the output format (**PNG**, **JPEG**, or **TIFF**).

`janela.config()` - Allows you to customize the window.

`janela.mainloop()` - Keep the window open and listen to the events.

```
janela = tk.Tk()
janela.title("Conversor de Imagens - Eliezer Carvalho")
janela.geometry("1200x1200")
#janela.iconbitmap(r"C:\Users\eliez\Desktop\upload_document
janela.config(bg = "#101820")
```

Image 5 - Example of using the ***Tkinter*** library

4. Pathlib

It is a modern **Python** library for manipulating file systems and directories in an object-oriented manner. It replaces the use of the traditional **OS** library in many scenarios.

Some of the functions used were:

`Path()` - Creates an object representing directories or files.

`iterdir()` - Itera over all files within the directory, allowing each image to be processed.

`saida.mkdir(exist_ok = True)` - Creates a valid folder. `exist_ok = True` prevents errors if a folder with the same name already exists.

```
pasta = Path(caminho)
```

Image 6 - Examples of using the ***Pathlib*** library

```
for ficheiro in pasta.iterdir():
    if ficheiro.suffix.lower() in extensões:
```

Image 7 - Examples of using the ***Pathlib*** library

Detailed Explanation of the Code

`def (converter_imagens)`

1. Creating an output folder

```
pasta_saida = pasta_original / f"Fotografias{formato_desejado.lower()}"

pasta_saida.mkdir(exist_ok=True)
```

2. Iteration over the files in the folder

```
for ficheiro in pasta_original.iterdir():
    if not ficheiro.is_file():
        continue
extensão = ficheiro.suffix.lower()
```

Checks each file and ignores subfolders, detecting the extension to determine whether it is **RAW** or a common format.

3. Handling of the most common formats (**JPG**, **PNG**, and **TIFF**)

Open the image with `Pillow`, then convert the color mode if necessary:

- **JPG → RGB**
- **PNG/TIFF → RGBA or RGB**

Save with maximum quality (95%) in the desired format and update the log with `log_widget.insert()`.

4. **RAW** Format Processing

Reads the **RAW** file with `raw.imread()` and performs post-processing with controlled parameters.

```
RGB = raw.postprocess(
   use_camera_wb = True,
      output_bps = 8,
  no_auto_bright = False,
    gamma = (2.2, 4.5),
output_color=rawpy.ColorSpace.sRGB
)
```

Next, convert the **NumPy** array to a **PIL** image (`Image.fromarray(RGB)`), change the color mode to be compatible with the output format, save, and update the log.

`def (selecionar_pasta)`

Allow the user to select the source folder through the graphical interface.

`def (tkinter)`

The main window (`Tk()`) sets the size, title, and background color.

The **main widgets** are: the “Select Folder” **button**, which calls the `select_folder()` function; the **input field**, which displays the path of the selected folder; the **label** and **combo box**, which allow you to select the output format (**JPG**, **PNG**, or **TIFF**); the “Convert” **button**, which calls the `converter_images()` function and passes the folder, format, and log.

Integration with functions:

Tkinter variables (`StringVar`) enable synchronization between the interface and backend functions.

The ***lambda*** used in the conversion button ensures the correct passing of parameters.

Challenges Encountered

Throughout the development of the project, I faced several technical and conceptual challenges that contributed to improving my understanding of the problem and the quality of the final solution.

The first significant challenge was understanding and working with the ***Tkinter*** library. Although it is a standard **Python** library, its approach to graphical interfaces is relatively old and differs greatly from modern programming structures. Creating an intuitive interface that allowed users to select folders, choose the output format, and track the conversion progress in real time required a detailed study of **widgets**, linked variables (`StringVar`), and methods for updating the interface without blocking.

Another important challenge was dealing with the conversion of images in different color modes. Many **RAW** images or images in formats such as **PNG** may have alpha channels or paletted palettes that are not compatible with the **JPG** format. It was necessary to implement mode checks (`image.mode`) and appropriate conversions, such as **RGBA** → **RGB** for **JPG** and **P** → **RGBA** for **PNG/TIFF**, in order to ensure that each image was converted correctly, without loss of quality or saving errors.

Bulk processing of **RAW** files also posed a technical challenge. Since each camera manufacturer uses different extensions (e.g., **.cr3**, **.nef**, **.arw**, and **.dng**) and different sensor standards, it was necessary to use the ***RawPy*** library and the correct post-processing configuration (white balance, gamma correction, and **sRGB** color space). Ensuring that the process was efficient, preserved quality, and could handle large folders without crashing the interface was one of the most critical aspects of the project.

Finally, the implementation of a robust batch conversion workflow that could scan entire folders, automatically create output directories, and log in real time for the user required attention to error handling, file system compatibility (**Windows, Linux**), and code modularity.

In short, all these challenges contributed to the development of skills in programming, image manipulation, and interface design, making the project functional, reliable, and scalable.

Results Achieved

The project resulted in an efficient, practical, and intuitive tool for the mass conversion of **RAW** images and common **JPG**, **PNG**, and **TIFF** formats.

The interface, built with ***Tkinter***, allows the user to quickly select a folder, choose the desired format, and perform the conversion in a safe and organized manner, without the risk of overwriting the original files. Each image is processed individually, with color and image mode compatibility checks performed to ensure that the final quality is preserved.

In the case of **RAW** images, the tool applies the appropriate post-processing, including camera white balance, gamma correction, and conversion to **sRGB**, producing compatible and visually consistent files. Common files are converted while maintaining fidelity, with only the strictly necessary adjustments made for compatibility with the output format.

Real-time logging allows you to track the progress of each file, informing you of completed conversions and any files that may have been skipped due to incompatibilities or errors, increasing the reliability and transparency of the process.

In addition, the project has demonstrated modularity and scalability, and can be extended to support new formats, apply additional filters, or integrate parallel processing without requiring major changes to the existing architecture.

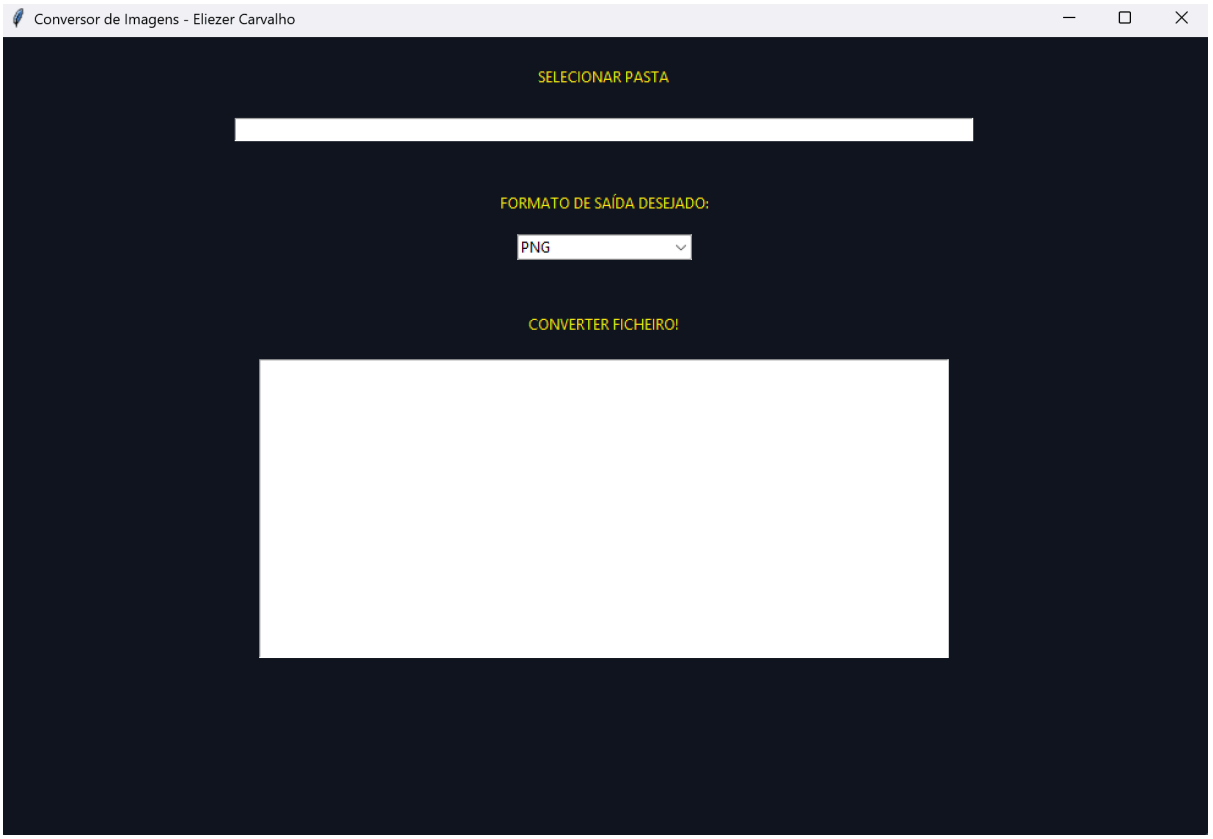

Image 8 - Final Result